

INTER-PROCESS COMMUNICATION ON A COMPUTER

Background of the Invention

Technical Field

5

This invention relates to inter-process communication in a computer environment. One embodiment of the invention relates to the inter-process communication on a single computer environment.

10

Description of Prior Art

15

Inter-process communications (IPC) use various communication methods, for example, shared memory, pipe, message queue or semaphore, to enable interactions between different application processes within the same computer or among different computers in a computer network system. The inter-process communications among multiple applications on a single computer allows individual application processes running on the computer to work together cooperatively.

20

25

The existing cross-platform inter-process communication methods, such as those based on the Common Object Request Broker Architecture (CORBA), can be used for the inter-process communication between application processes running on different computers with different operating systems. For example, CORBA provides a set of common interfaces through which object-oriented software can communicate, regardless of the computer platform on which the object-oriented software is running. However, such cross-platform inter-process communication methods are too heavy-weight to be a good solution for application processes running on a single computer. Existing inter-process communication methods for

application processes running on a single computer, such as Component Object Model (COM) from Microsoft, Inc., are operating system-dependent and are not compatible with other platforms. For example, it would be difficult to port a set of software programs based on Component Object Model (COM) running on operating systems provided by Microsoft, Inc. to other operating systems, such as UNIX or Mac OS, etc.

Therefore, it would be advantageous to provide a method of inter-process communication for processes running on a single computer so that software applications based on such a method can be easily implemented and ported to different computer platforms running different operating systems.

Summary of Invention

The invention provides a method of inter-process communication between at least two application processes on one computer. One embodiment of the invention includes a first process of a first application for determining a name of a first file in a file system of the computer. The name of the first file is associated with a second application. The first file contains information for the first process to connect to a second process of the second application for inter-process communication. The first process initiates a first connection to the second process using the information contained in the first file. The first process communicates with the second process using the first connection if the first connection is successfully established; and the first process starts a third process of the second application if the first process fails to establish a connection with the second process.

The invention further allows the first process to initiate a second connection to the third process using the information in the first file, in response to the third process. This informs the first process that the third process is ready for a connection. The third process is started in a server mode without a user interface. Further, the first process fails to establish a connection with the second process because the second process is not running. The first file being missing from the file system indicates that the second process is not running.

In one embodiment, when the first process is started, the first process determines if a fourth process of the first application is running. The first process requests the fourth process to perform a task for the first process if the fourth process is running; and the first process exits after requesting the fourth process to perform the task for the first process.

Furthermore, the first process determines if the fourth process of the first application is running. This includes the first process of the first application determining a name of a second file in the file system of the computer, the name of the second file being associated with the first application. The second file being missing from the file system indicates that the fourth process of the first application is not running.

The second file contains information for the first process to connect to a fourth process for inter-process communication. Failure in connecting to the fourth process using the information contained in the second file indicates that the fourth process of the first application is not running; and success in connecting to the fourth process using the information contained in the second file indicates that the fourth process of the first application is running.

The first process communicates with the second process using the first connection through an Application Program Interface (API), which is platform independent. When the second process is started, the second process determines if a fourth
5 process of the second application is running.

In one embodiment, the second process requests the fourth process to perform a task for the second process if the fourth process is running; and the second process exits after requesting the fourth process to perform the task for the second process.

10

Brief Description of The Drawings

FIG. 1 is a block schematic diagram showing two application processes in inter-process communication on a single computer according to one embodiment of the invention;

15

FIG. 2 is a figure diagram showing a method of inter-process communication between two applications on a single computer according to the invention;

FIG. 3 is a figure diagram showing a method of inter-process communication
20 between two processes of an application program on a single computer according to one embodiment of the invention;

FIG. 4 is a figure diagram showing a method of inter-process communication between two applications on a single computer according to the invention;

25

FIG. 5 is a figure diagram showing a method of inter-process communication between two applications on a single computer according to the invention.

5

Detailed Description of the Invention

The IPC process according to the one embodiment of the invention provides a lightweight, cross-platform mechanism for applications to communicate with one another. For example, the IPC mechanism according to one embodiment of the invention is purposefully restricted to communication between applications running on a single computer under the control of a single instance of an operating system.

In addition, a mechanism is used to ensure that only one instance of each application runs at a time to provide the service. For example, when a second instance of an application is started while a first instance of the application is still running, the second instance of the application communicates the task to be performed to the first instance of the application. After the second instance of the application communicates the task to the first instance, the second instance automatically exits. The task is then performed by the first instance of the application. More details are described below.

FIG. 1 is a block schematic diagram showing two application processes in inter-process communication on a single computer according to one embodiment of the invention. Referring now to FIG. 1, a computer 102 includes an operating system 110. The operating system provides basic communication facilities for the application processes running under its control. These communication facilities are typically platform-dependent. The facilities available in one type of operating system may not be available or suitable for use in another type of operating system.

However, various different types of operating systems all provide access to files. In one embodiment of the invention, an inter-process communication mechanism is based on the access to rendezvous files, which contain platform- dependent information for establishing inter-process communication. Those skilled in the art will appreciate that a product other than rendezvous may be used to implement the discussed invention. The IPC library 112 makes use of the available communication facilities provided by the operating system 110 to provide an inter-process communication mechanism through a cross-platform Application Programming Interface (API) for different application processes, such as a process A 106 and a process B 104. Because the application programs for the process A 106 and the process B 104 are written using the cross-platform API for inter-process communication, the application programs can be easily ported to an operating system that is different from the operating system 110.

In one embodiment of the invention, each application program has an associated rendezvous file. The rendezvous file contains the information about a running instance of the application program so that other running process can use the information to establish inter-process communication with this running instance of the application program. For example, a client application process, *e.g.* the process A 106, can find a server application process, *e.g.* the process B 104, by mapping the server's name to a file, *e.g.* the rendezvous file A 116, in the file system 118.

In one example, the rendezvous file of an application program is named after the application program and placed in a predetermined location in the file system so that other application processes can locate the rendezvous file without prior knowledge of the running instance of the application program. The file contains rendezvous

information, including parameters necessary for the client to connect to the server. For example, the file system 118 can have a rendezvous file A 114 for process A of an application program, *e.g.* Application A, and a rendezvous file 116 for process B of another application program, *e.g.* Application B. The file 114 contains rendezvous information for the application process 104 and the file 116 contains the rendezvous information for the application process 108.

When the application process 104 needs to establish an inter-process communication channel to communicate with the application process A 106, the application process 104 computes the file name of rendezvous file A 116 from the name of the application program of process 106, opens the rendezvous file A 116, connects to the application process 106 using the information in the rendezvous file A 116, and communicates with process 106, using IPC library 112 through platform-independent API 108.

If a server application, *e.g.* process 106, is not running, the rendezvous file 116 will be missing or contain stale information that causes the connection to fail. In that case, a client application process 104 can start the server process 106, with a command line argument of "-server," which tells the server application to initialize itself and to get ready to serve the IPC requests, but not to display a user interface (UI). The client application process can start the server process in a server mode without a user interface so that the user is not bothered with a user interface of the server application program while the client application process obtains service from the server application. After the server process, *e.g.* 106, has initialized itself, the server process informs the client process, *e.g.* 104, that the server 106 is ready for

IPC traffic and the client 104 makes a second attempt to connect, which normally succeeds.

In one embodiment of the invention, application processes can use the rendezvous files to combine multiple instances of an application program into one running instance. For example, when the application process B 104 of an application program is started to perform a task, the application process B 104 uses the IPC mechanism to look for a prior instance of the same application program. For example, before the application process B 104 initializes its rendezvous file, the application process 104 can try to locate the rendezvous file of the same application program and to communicate with a prior instance of the application program if the rendezvous file exists. If a prior instance 106 exists, the application process B 104 can establish an inter-process communication channel with the prior instance 106 and communicates the task of the application process B 104, such as the command line arguments for the application process, to the prior instance 106. After instructing the prior instance 106 to perform the task of the application process B 104, the application process B 104 exits. In this way, the action requested by the user of the second instance of an application program is actually performed by the first instance of the application program. Thus, multiple instances of a same application program can consolidate the tasks to be performed in one instance of the application program.

In one embodiment of the invention, all instances of one application program use the same rendezvous file. The rendezvous file contains valid information for only one instance of the application program at a time. When tasks for multiple instances of an application program are consolidated for process in only one instance of the

application program, the computational resources of the computer can be used more efficiently. Further, inter-process communication between different instances of a same application or different applications can be simplified.

5 The client and the server status of application processes 104 and 106 can be interchangeable. For example, the process 104 can be an instance of a first application program, *e.g.* an email program, which has rendezvous information stored in file 116; and, the process 106 can be an instance of a second application program, *e.g.* an address book program, which has rendezvous information stored in
10 file 116. The process 106 of the address book program can initiate an IPC channel using the rendezvous file 114 and request the process 116 of the email program to send an email message. In this scenario, the process 106 of the address book program is a client process and the process 104 of the email program is a server process.

15

In another scenario, the process 104 of the email program can initiate an IPC channel using the rendezvous file 116 and request the process 106 of the address book program to look up an email address of a person, in which the process 106 of the address book program is a server process and the process 104 of the email
20 program is a client process. In general, a set of application programs may each provide services to others and obtain services from others. The set of application program can use the method of IPC according to embodiments of the invention to communicate with each other.

25 One embodiment of the invention provides a cross-platform API 108 for exchanging messages between applications. Each message has one or more key and value

pairs. The applications specify the meanings of the keys and values so that applications can exchange meaningful information. There are two basic forms of messages: 1) commands from one application to tell another application to do something; and 2) events sent from a source application to a target application to
5 inform the target application that something of interest has occurred in the source application.

In one embodiment of the invention, the cross-platform API 108 provides a simple mechanism for transmitting sets of keys and values between two processes running
10 on the same computer. In one implementation, the keys and values are transmitted as strings; and, the unit of the transmission of the keys and values is an IPCMessage, which includes a set of keys and values.

For example, the following code creates an IPCMessage and associates some keys
15 and values with it. In the following code example, the message "item" has keys "cn," "mail," "locality," and "st," which may represent a name, an email address, locality, and a state. Key "cn" has a value of "Roger" in the message; key "mail" has a value of "rogc@netscape.com;" key "locality" has a value of "Granite Bay"; and key "st" has a value of "CA."

20

```
IPCMessage* item = IPC_NewMessage();  
IPC_MessageSetValue(item, "cn", "Roger");  
IPC_MessageSetValue(item, "mail", "rogc@example.com");  
IPC_MessageSetValue(item, "locality", "Granite Bay");  
25 IPC_MessageSetValue(item, "st", "CA");
```

More complex data structures can be stored in an IPCMessage by using the function "IPC_Serialize." In the following code example, the item that has key "cn" with value "Roger" and key "mail" with value "rogc@example.com" is serialized as the value of key "item1;" and, the item that has key "cn" with value " Bob" and key "mail" with value " bob@example.com" is serialized as the value of key "item2." The message includes keys "item1" and "item2" with the corresponding serialized values.

```
IPCMessage* message = IPC_NewMessage();
IPCMessage* item = IPC_NewMessage();
10 IPC_MessageSetValue(item, "cn", "Roger");
IPC_MessageSetValue(item, "mail", "rogc@example.com");
IPC_MessageSetValue(message, "item1", IPC_Serialize(item));
IPC_DestroyMessage(item);
item = IPC_NewMessage();
15 IPC_MessageSetValue(item, "cn", "Bob");
IPC_MessageSetValue(item, "mail", "bob@example.com");
IPC_MessageSetValue(message, "item2", IPC_Serialize(item));
IPC_DestroyMessage(item);
```

20 The preceding example creates a message having two keys: "item1" and "item2." To extract the nested keys and values that are serialized as the values of the two keys, the receiving code can use IPC_Deserialize, as illustrated in the following code example. In the following code example, the received message is deserialized as item1 and item2, respectively. The values for keys "cn" and "mail" for item1 is
25 retrieved and stored in variables name1 and mail1, respectively; and the values for keys "cn" and "mail" for item2 is retrieved and stored in variables name2 and mail2, respectively:

```

IPCMessage* item1 = IPC_NewMessage();
IPC_Deserialize(item1, IPC_MessageGetValue(message, "item1"));
const char* name1 = IPC_MessageGetValue(item1, "cn");
const char* mail1 = IPC_MessageGetValue(item1, "mail");
5 IPCMessage* item2 = IPC_NewMessage();
IPC_Deserialize(item2, IPC_MessageGetValue(message, "item2"));
const char* name2 = IPC_MessageGetValue(item2, "cn");
const char* mail2 = IPC_MessageGetValue(item2, "mail");

```

10 FIG. 2 is a flow diagram showing a method of inter-process communication between two applications on a single computer according to the invention.

Referring now to FIG. 2, a client process determines a name of a file associated with a target application 202. If the file associated with the target application does not
 15 exist, the client process may abort the attempt to connect to the target application 204. If the file associated with the target application exists, the client process initiates a connection to the target application using the information containing in the file 206.

It is then determined whether the connection is successful or not 208. If it is
 20 determined that the connection is not successful 208, the information in the rendezvous file may be invalid, *stale*, and the client process aborts the attempts to connect to the target application 204. If it is determined that the connection is successful 208, the client process communicates with the target application using the established connection 210.

25

The client process may start one instance of the target application, which may display with or without a user interface. Typically, however, the client process starts the target application in the server mode without displaying a user interface.

Once an instance of the target application is started, the target application creates its associated rendezvous file. The client process may wait for the target process to create the rendezvous file. Alternatively, the client process may wait for the target process to open a channel using the rendezvous file of the client process and to
5 inform the client process that the target process is ready for IPC traffic.

Once the client process locates the rendezvous file of the target application, the client process starts connecting to the target application process by opening an IPC communication channel using the information in the rendezvous file.
10

In one embodiment of the invention, each member of a group of IPC applications according to the invention provides services to the others. For example, the address book provides a service for resolving addresses and the mailer provides a service for creating new compose windows.
15

In one implementation, the IPC process according to the invention is provided by a cross-platform C API, a programming language which includes codes in a directory, *e.g.* the lib/ipc directory of the source tree. Each application using the IPC process subclasses a base class, *e.g.* PhotonApp from lib/wxutil. PhotonApp uses lib/ipc to
20 establish itself as a server to other applications by creating the rendezvous file, and also enforces the invariant that only one instance of each application runs to perform tasks at a time. Other started instances transfer their tasks to the running instance for which the information in the rendezvous files is valid. The details are described below.
25

FIG. 3 is a flow diagram showing a method of inter-process communication between two processes of an application program on a single computer according to one embodiment of the invention.

5 Referring now to FIG. 3, a first process in the application program is started and its associated rendezvous file is created subsequently 302. The rendezvous file describes a communication channel that allows a later started processes to connect to the first process. Then, a second process of the same application program is started 304.

10

The second process of the application program determines whether a prior instance of the application program exists or not, *e.g.* another process of the application program is already running or not 306. If the rendezvous file of the application program already exists, the second process thus determines that another process,
15 *i.e.* the first process, is already running. The second process of the same application program can use the information in the rendezvous file to connect to the first process.

If it is determined that a prior instance of the application program, *e.g.* the first
20 process is already running 306, the second process then requests the first process to perform the task of the second process using the communication channel 306. Then, the second process automatically exits 310. Thus, the task of the second process is actually performed by the first process.

25 However, if the rendezvous file of the application program does not exist or the connection using the rendezvous file of the application program fails, no prior

instance of the application program is running. Thus, the second process of the application creates the rendezvous file and/or stores the rendezvous information in the rendezvous file of the application program 312.

- 5 In this way, the invention provides a method of IPC where only one copy of an application runs to perform tasks at a given moment. It is understood that an instance of an application program is a copy of running process of the application program.
- 10 FIG. 4 is a flow diagram showing a method of inter-process communication between two applications on a single computer according to the invention.

Referring now to FIG. 4, the invention provides a method of IPC between two applications on a single computer using a channel. To transmit an IPCMessage
15 from one process to another an IPCChannel is used. One process is the server and the other process is the client. For example, the client creates an IPCChannel using
IPC_OpenChannel: IPCChannel* chan =
IPC_OpenChannel("nsmail").

- 20 The client uses IPC_WriteMessage to transmit a message to the server and
IPC_ReadMessage to receive a response:
IPCChannel*chan = IPC_OpenChannel("nsmail");
if (chan){
IPCMessage* message = IPC_NewMessage();
25 IPC_MessageSetValue(message, "ph_command", "compose");
IPC_MessageSetValue(message, "to", "roger@example.com");
if (IPC_WriteMessage(chan, message) == PR_SUCCESS) { IPCMessge*
reply=IPC_NewMessage();

```

    if (IPC_ReadMessage(chan,reply,
PR_INTERVAL_NOTIMEOUT)==PR_SUCCESS){/*
    * mailer launched a compose window with
    * "roger@example.com in the "To:" field.
5    */}
    IPC_DestroyMessage(reply);
    IPC_DestroyMessage(message);
    IPC_CloseChannel(chan);

```

- 10 `IPC_OpenChannel` takes the name of an application and returns an `IPCChannel` that can be used to communicate with the one and only instance of that application.

The client finds the server using a file in the preference directory 402. The file is associated with the server. The client determines whether the rendezvous
15 information is present in the server 404. If it is determined that the rendezvous information is present in the server 404, the client reads the rendezvous information and tries to connect to the server 406.

For example, when the string `nsmail` is passed to the interface channel
20 `IPC_OpenChannel`, the rendezvous file is `$PREFS_DIR/nsmail.sok`. The "nsmail.sok" is a file that contains a platform-dependent information that a client can use to contact a server.

If "nsmail.sok" is present, `IPC_OpenChannel` reads it and then tries to establish a
25 connection to the 16-bit port number specified using the localhost interface to the server using the platform-dependent parameters in `nsmail.sok`.

The server then determines whether a connection is successfully established 408. If it is determined that the connection is successful 408, the client and the server establish a communication channel 410. For example, If a connection is established, the client writes the client key to the connection and then reads 64 bits from the connection. If the read is successful and the 64 bits read from the connection match the 64-bit server key, an IPCChannel is created and returned to the caller.

If any of the above steps fail, the client assumes that the server is not running and should be started. The client starts the server 412 and waits for the server to be initialized 414. For example, the NSPR function CreateProcess is used to start the server, and then a platform-specific mechanism is used to wait for the server to initialize, e.g. WaitForInputIdle on Windows and Inheritable file descriptor on other platforms.

After the server has been initialized 412, 414, the client attempts to connect to the server again 416. For example, the client may attempt to connect using nsmail.sok as in the preceding paragraph. If the second attempt to connect fails, IPC_OpenChannel returns NULL to the caller 418.

FIG. 5 is a flow diagram showing a method of initiating an inter-process communication between two applications on a single computer according to the invention. The invention provides an efficient method of making connections between two applications via an IPC process to ensure that only one instance is running.

Referring now to FIG. 5, the server initialization code is stored in a file on the server. For example, a server initialization code may be stored in a file "lib/wxutil/PhotonApp.cpp". At initialization, the server acquires a semaphore 502, e.g., an NSPR PRSem semaphore to prevent races involving multiple clients
5 launching the same server simultaneously. After acquiring the semaphore 502, the server determines whether a prior instance exists 504.

If it is determined a prior instance exists 504, it is further determined whether a connection to the prior instance can be established 506. If such a connection can be
10 established 506, the server allows the prior instance to run and exits 508. For example, after a server sends its command line arguments to the previously existing instance, the server releases the semaphore and exits.

If it is determined that no previous instance exists 504, the server generates a rendezvous file 510. A typical rendezvous file contains a client key, a server key and
15 transport parameters information. For example, the server creates a socket and binds it to a port on the local host interface. Then the server generates two 64-bit random numbers, and writes out a rendezvous file, *e.g.* a rendezvous file for Mail can be "nsmail.sok."

20 The server waits for connections from a client 512. Typically, a new thread is spawned to await connections from the client. The client attempts to establish a connection to the server. In one embodiment, the client reads the rendezvous file and starts sending a client key to the server. The server verifies the client key. Meanwhile, the server sends to the client the server key and the client verifies the
25 server key. It is then determined whether the server is successfully connected to the client 514.

If it is determined that the server is successfully connected to the client 514, it is then determined whether the user interface on the server shall be displayed 516.

- 5 If it is determined that the user interface on the server shall be initialized 516, in one embodiment, the semaphore is released and the server's user interface initializes 518. Alternatively, if it is determined that the user interface on the server shall not be initialized 516, the server stays hidden with the user interface hidden 520. For example, the server can be started with the "-server" argument.

10

After the client and the server are successfully connected and properly initiated, an IPC process begins 522. Typically, an IPC Message traffic starts to flow.

15

If it is determined that the server is not successfully connected to the client 514, the client initiates the server 524.

In this way, one embodiment of the invention provides a method of initiating a server application and only one copy of the server application by a client application running on a single computer.

20

Although the invention is described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the claims
25 included below.